Deliverable No. 7.3

Demonstration of system interface

Grant Agreement No.:	270089
Deliverable No.:	D7.3
Deliverable Name:	Demonstration of system interface
Contractual Submission Date:	31/07/2013
Actual Submission Date:	27/08/2013

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	***p-medicine***
Project Full Name:	From data sharing and integration via VPH models to personalized medicine
Deliverable No.:	D 7.3
Document name:	Demonstration of system interface
Nature (R, P, D, O)[1]	D
Dissemination Level (PU, PP, RE, CO)[2]	RE
Version:	2
Actual Submission Date:	27/08/2013
Editor: Institution: E-Mail:	Benjamin Jefferys UCL b.jefferys@ucl.ac.uk

ABSTRACT:

This deliverable describes the programmatic and user interfaces to the data warehouse, developed in Task 7.5.2

KEYWORD LIST: REST, HTTP, programmatic interface, user interface, portlet, portal, web interface, data warehouse

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 270089.

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

MODIFICATION CONTROL			
Version	Date	Status	Author
1.0	6/08/2013	Draft	Benjamin Jefferys
2.0	27/08/2013	Final	Benjamin Jefferys

List of contributors

- Benjamin Jefferys, UCL
- Giorgos Zacharioudakis, FORTH (reviewer)

Contents

1 Executive Summary

This document describes the programmatic and user interfaces to the data warehouse, in order to demonstrate them. The programmatic interface is a RESTful HTTP interface based upon resources representing files and a triplestore, using the available HTTP verbs to access and alter these resources. Some additions are made to the normal HTTP interface to provide the required functionality, in particular for indicating that one file supersedes another. This interface is now deployed at `http://eh-services.vph.psnc.pl:8282`. The user interface is a Liferay portlet providing free-text and SPARQL searching, and faceted browsing and filtering of results, as well as links to documentation and news about the warehouse, and a list of the latest changes for auditing purposes. This is as yet only deployed at the UCL internal instance of Liferay.

2 Programmatic interface

As a RESTful interface, the specification below is in addition to the HTTP 1.1 specification[3].

This interface is now deployed at `http://eh-services.vph.psnc.pl:8282`.

Request headers

One request header is common to all calls to the data warehouse:

`X-Session-Auth` An identifying string used for authenticating the requesting user for this session. This comes from the single-sign-on service. This may change name in the future to come into line with other p-medicine RESTful services. The precise nature of this string is dependent upon the security infrastructure. It is opaque to the warehouse and simply passed to the security infrastructure for verification.

Where a request header must specify an RDF file format, the following formats are supported:

- `application/rdf+xml`
- `text/plain` (N-Triples)
- `application/x-binary-rdf` (Sesame Binary RDF)

Transactional update format

In order to apply an atomic change to the triplestore consisting of addition and deletion of many triples in one request, a POST request to a triplestore statements resource can include a body which describes these in a simple extension to the text/plain N-Triples[4] based RDF file format.

Briefly, each line of an N-Triples file consists of three elements, separated by spaces:

`subject predicate object`

Where one of these elements includes a space, it must be surrounded by double quotations marks. The `application/x-pmed-triplestore-transaction` adds to this:

`action subject predicate object`

Here, *action* is one of P or D, where P indicates the triple on that line should be created (equivalent to a POST method with a normal RDF file format) and D indicates the triple on that line should be deleted (equivalent to a DELETE method). The change will be applied to the triplestore atomically, and the transaction will be regarded as a single unit in terms of logging, provenance, and history.

Resources and methods

2.1.1 /v1/triplestore

Resource for accessing triples in the main data warehouse triplestore. This interface is based upon the OpenRDF Sesame RESTful interface.

[3] http://www.w3.org/Protocols/rfc2616/rfc2616.html

[4] http://www.w3.org/2001/sw/RDFCore/ntriples/

2.1.1.1 GET

Preferred method for executing a SPARQL query on the main data warehouse triplestore. Parameters:

`query`	URL-encoded query to evaluate
`timeout`	Optional limit in whole seconds on evaluation time of query. If omitted, query time is unlimited

Request headers:

`X-Session-Auth`	Described above
`Accept`	Requested mimetypes for response. For CONSTRUCT queries, valid types are any of the RDF results types described in the "Request headers" section above. For ASK queries, valid values are *`application/sparql-results+xml`* or *`text/boolean`* (string *true* or *false*). For SELECT queries, valid values are *`application/sparql-results+xml`*, *`application/sparql-results+json`*, *`application/x-binary-rdf-results-table`*, *`text/csv`*, *`text/tab-separated-values`*

Comma- and tab-separated value responses for SELECT queries are constructed with the binding variable as the column names in the first row, followed by a row per binding result.

2.1.1.2 POST

Method for executing a SPARQL query on the main data warehouse triplestore when the query is considered too long to transmit as part of the request URL.

Additional request headers:

`Content-Type`	`application/x-www-form-urlencoded`

The body should contain the variables described for the GET method above, in www-form-urlencoded format.

2.1.2 /v1/triplestore/version

Resource representing the current version number of the triplestore. This is incremented after every transaction which alters the triplestore, and can be used in the future to access previous versions.

GET

Get current version number and date and time of last change to the triplestore.

Request headers:

`X-Session-Auth`	Described above

Reponse headers:

`X-Version`	Integer version number of the triplestore
`Last-Modified`	HTTP-date of the last change to the triplestore, when this version was created.

2.1.3 /v1/triplestore/statements

Resource representing all the statements (triples) in the main data warehouse triplestore.

2.1.3.1 GET

Fetches statements from the repository.

Parameters:

`subj`	Optional subject which all returned statements should have
`pred`	Optional predicate which all returned statements should have
`obj`	Optional object which all returned statements should have

Request headers:

`X-Session-Auth`	Described above
`Accept`	Any of the RDF results types described in the “Request headers” section above

2.1.3.2 POST

Performs updates on the triplestore, either by adding triples from an RDF document provided in the request, or by executing a SPARQL 1.1 Update query provided in the request.

Parameters:

`update`	Optional SPARQL 1.1 update string to be executed. Note that this parameter may be part of the request URL, or in the body of the request with a Content-Type `application/x-www-form-urlencoded`
`baseURI`	Optional base URI to resolve any relative URIs found in uploaded data against.

Request headers:

`X-Session-Auth`	Described above
`Content-Type`	Where RDF statements are provided for addition to the triplestore, this should be any of the RDF results types described in the

	"Request headers" section above. Where a SPARQL 1.1 Update is sent in the body, this should be *application/x-www-form-urlencoded*.

Response headers:

X-Transaction	A URL representing this transaction, which is used to refer to this transaction in the logging, provenance and history metadata model.

2.1.3.3 PUT

Performs updates on the triplestore by adding triples from an RDF document provided in the request.

Parameters:

baseURI	Optional base URI to resolve any relative URIs found in uploaded data against.

Request headers:

X-Session-Auth	Described above
Content-Type	Any of the RDF results types described in the "Request headers" section above.

Response headers:

X-Transaction	A URL representing this transaction, which is used to refer to this transaction in the logging, provenance and history metadata model.

2.1.3.4 DELETE

Deletes triples from the triplestore. As a precaution to prevent the extremely rare case that a user wants to delete all the statements in the repository, or all statements with a particular object, subject or predicate, normal users are required to specify at least two of *subj*, *pred* or *obj*. Extension of permissions for broader delete commands can be applied through the authorisation system.

Parameters:

subj	Optional subject which all deleted statements should have
pred	Optional predicate which all deleted statements should have
obj	Optional object which all deleted statements should have

Request headers:

`X-Session-Auth`	Described above
`Content-Type`	Any of the RDF results types described in the “Request headers” section above.

Response headers:

`X-Transaction`	A URL representing this transaction, which is used to refer to this transaction in the logging, provenance and history metadata model.

2.1.4 /v1/triplestore/versions/*versionNumber*

Resource representing a particular version of the triplestore. Applicable methods and method signatures are identical to `/v1/triplestore`. Note that previous versions of the triplestore are not maintained permanently, and constructing a previous version may take some time. Making a request to a particular version of the triplestore triggers the construction of that version, if it does not already exist. If this happens, the server will immediately return a 503 (Service Unavailable) response, with the following response header:

`Retry-After`	HTTP-date or number of seconds after which to retry the request. Note that a retried request may still fail with a 503 response and a further Retry-After time.

2.1.5 /v1/triplestore/versions/*versionNumber*/statements

See `/v1/triplestore/versions/versionNumber`: this is a resource representing statements in a particular version of the triplestore. Its interface is identical to `/triplestore/statements`, *except* that PUT, POST and DELETE methods are not available. Previous versions of the triplestore are immutable.

2.1.6 /v1/filestore

Resource representing the contents of the filestore.

2.1.6.1 POST

Upload a new file to the filestore.

Request headers:

`X-Session-Auth`	Described above
`Content-Type`	Mime-type of uploaded file
`Content-Location`	Optional absolute or relative URI for source of uploaded file. This is used for informational purposes only during subsequent

	annotation, and may not be available due to legal constraints upon what data is stored in the warehouse.
`Last-Modified`	Optional date and time (HTTP-date) of last modification of file contents.
`X-Supercedes`	URL of a file which this file supercedes

If the file is successfully uploaded and stored, the response code 201 (Created) will be returned with response headers including:

`Location`	Complete URL for uploaded file. The name of the file is generated internally and is universally unique.

2.1.7 /v1/filestore/*fileName*

2.1.7.1 GET

Retrieves the specified file.

Request headers:

`X-Session-Auth`	Described above

2.1.7.2 DELETE

Deletes the specified file. Deleting a file should only be done if it contains data which should not be present in the warehouse - for example, it has serious errors or it contains personally identifiable information. Normally, if a file is to be replaced with a newer version (for example, with additional data), the new file should be uploaded through a POST method on `/filestore`, with the `X-Supercedes` header set to the URL of the file being replaced. This is so that analyses which rely upon the previous versions of the file still work.

Request headers:

`X-Session-Auth`	Described above

2.1.8 /v1/filestore/*fileName*/triples

Resource representing the metadata triples extracted from the file upon submission.

2.1.8.1 GET

Retrieve the triples file, in N-triples format.

2.1.9 /v1/filestore/*fileName*/ontology

Resource representing the ontology extracted from the file upon submission, and used in `/v1/filestore/`*`fileName`*`/triples`.

2.1.9.1 GET

Retrieve the ontology file, in OWL format.

2.1.10 /v1/filestore/*fileName*/annotationDescription

Resource representing a file which describes how to transform the triples in `/v1/filestore/`*`fileName`*`/triples` into triples which can be added to the data warehouse. The format of this file and the nature of this process are to be developed by UPM.

2.1.10.1 GET

Retrieve the annotation description file.

2.1.10.2 PUT

Upload the annotation description file. This process triggers transformation of the raw graph into new triples for the triplestore. Triples which are added as a result of this process are recorded internally. If the annotation description subsequently changes, the previously added triples are removed prior to addition of the new triples.

2.1.11 /v1/imagestore

Resource for basic access to images in the DICOM server. This interface is not based upon the WADO-RS interface in the DICOM standard, since such an interface is provided directly by the DICOM server. It is intended as a very simple method for getting an image via a standard HTTP request from a workflow engine or other platform which may not support DICOM protocols. If a use case is later defined which requires it, this resource may be extended to support upload and deletion of images via POST and DELETE methods.

2.1.12 /v1/imagestore/*objectUID*

2.1.12.1 GET

Retrieves the specified file.

Request headers:

`X-Session-Auth`	Described above

2.1.13 /v1/imagestore/*objectUID*/triples

Resource representing the metadata triples extracted from the image upon submission.

2.1.13.1 GET

Retrieve the triples file, in N-triples format.

2.1.14 /v1/imagestore/*objectUID*/ontology

Resource representing the ontology extracted from the image upon submission, and used in `/v1/imagestore/`*`objectUID`*`/triples`.

2.1.14.1 GET

Retrieve the ontology file, in OWL format.

2.1.15 /v1/imagestore/*objectUID*/annotationDescription

Resource representing a file which describes how to transform the triples in `/v1/imagestore/objectUID/triples` into triples which can be added to the data warehouse. The format of this file and the nature of this process are to be developed by UPM.

2.1.15.1 GET

Retrieve the annotation description file.

2.1.15.2 PUT

Upload the annotation description file. This process triggers transformation of the raw graph into new triples for the triplestore. Triples which are added as a result of this process are recorded internally. If the annotation description subsequently changes, the previously added triples are removed prior to addition of the new triples.

Extracting raw triples from some common data sources

Three key data sources are CSV files, table-based relational and non-relational databases, and DICOM images (and similar filetypes containing metadata). In each case, the files will be parsed upon submission to the filestore or imagestore, and RDF triples will be extracted representing the contents of the file. This will transform the data into a common format for later annotation with tools developed by UPM.

For each file, a unique ontology will be created which reflects the original structure of the imported data. This is accessible through the programmatic interface. Additionally, there is a small ontology for terms common to many types of raw graph.

The following prefixes are used below:

f: `/v1/triplestore/rawgraphs/filestore/xxx/`

p: `/v1/triplestore/rawgraphs/ontology#` (the ontology for rawgraphs)

rdf: `http://www.w3.org/1999/02/22-rdf-syntax-ns`

hd: Prefix for p-medicine hdot ontology

2.1.16 CSV

Consider submitted CSV file `/filestore/xxx` with a first line that is determined to be a set of labels, such as:

“Label A”	**“Label B”**	**“Label C”**
2003	j	1
1999	q	2
2001	y	3

This will be transformed into RDF triples as follows:

```
f:row0          rdf:type                    p:row
f:row0          p:rowIndex                  0
f:row0          f:Label%20A                 "2003"
f:row0          f:Label%20B                 "j"
f:row0          f:Label%20C                 "1"
f:row1          rdf:type                    p:row
f:row1          p:rowIndex                  1
f:row1          f:Label%20A                 "1999"
f:row1          f:Label%20B                 "q"
f:row1          f:Label%20C                 "2"
f:row2          rdf:type                    p:row
f:row2          p:rowIndex                  2
f:row2          f:Label%20A                 "2001"
f:row2          f:Label%20B                 "y"
f:row2          f:Label%20C                 "3"
```

That is, a row is realised as an entity `f:rowX` of type `p:row`, with the row index and the data from the row related to that.

Since reliable type information is not available, everything will be interpreted as a string: conversion to other types is left for the user-directed annotation stage.

Where the first row is not determined to be a set of labels, the predicates `f:Label%20A`, `f:Label%20B` and `f:Label%20C` above will be replaced with `f:col0`, `f:col1`, `f:col2`, the number representing the column number of that data item.

2.1.17 Table-based relational and non-relational database files and dumps

2.1.17.1 Single table

This is a surprisingly common case for small trials. Consider submitted file containing a single table called "Table X", with the following contents and data types (*in italics*):

“Label A” ***date***	**“Label B”** ***string***	**“Label C”** ***integer***
2003	j	1
1999	q	2
2001	y	3

This will be transformed into RDF triples as follows:

```
f:Table%20X       rdf:type         p:table
f:Table%20X       p:name           “Table X”
f:Table%20X0      rdf:type         f:Table%20X
f:Table%20X0      p:rowIndex       0
f:Table%20X0      f:Label%20A      “2003”^^xsd:dateTime
f:Table%20X0      f:Label%20B      “j”
f:Table%20X0      f:Label%20C      1
f:Table%20X1      rdf:type         f:Table%20X
f:Table%20X1      p:rowIndex       1
f:Table%20X1      f:Label%20A      “1999”^^xsd:dateTime
f:Table%20X1      f:Label%20B      “q”
f:Table%20X1      f:Label%20C      2
f:Table%20X2      rdf:type         f:Table%20X
f:Table%20X2      p:rowIndex       2
f:Table%20X2      f:Label%20A      “2001”^^xsd:dateTime
f:Table%20X2      f:Label%20B      “y”
f:Table%20X2      f:Label%20C      3
```

This is essentially similar to CSV import, with some additions:

- An entity is created to represent the table
- The name of a table is interpreted as being the “type” of its rows

- The data types of the columns is used to appropriately type the data in the triplestore. This is done conservatively, resorting to string representation where conversion may be unreliable, and the details of how it is done will be particular to the data source.

The parts of the table schema which relate to integrity and data structure rules for relational databases (such as allowing NULL values, allowable values for certain columns, keys, and so on) are *not* included in the generated ontology or data structures. Binary objects will be stored as new files in the filestore, and referred to by URL.

2.1.17.2 Multiple tables

Importing multiple tables in general will proceed as for importing single tables. An important addition, however, is where table keys are described as linking two tables together. Where possible, this link will be expressed as a relation between rows. For example, if a foreign key is defined which links “Label A” and “Label B” in “Table X” to “Label M” and “Label N” in “Table Y”, a corresponding ontology entry will be created `f:foreignKeyIdentifier`. This will be used as a predicate to relate rows which are linked by the foreign key; for example, the following relates row 0 of Table X to row 15 of Table Y:

```
f:Table%20X0     f:foreignKeyIdentifier              f:Table%20Y15
```

Multiple `f:foreignKeyIdentifiers` can be created for each database.

2.1.18 DICOM images and other files containing defined metadata

DICOM image files can contain a large amount of simple key-value type metadata, as can many other file formats (images, microarray data, high throughput sequencing data, and so on). These are specified in a domain-dependent way. For example, DICOM files store metadata as extra Data Elements with numerical tags which are only meaningful in the context of the DICOM specification (for example, tag (0008,002A) represents the acquisition date and time of an image).

Such metadata should be translated into RDF triples that make use of the p-medicine ontology or other relevant ontologies where possible. For example a DICOM file `/v1/imagestore/xxx` with width 400, height 300, acquired 2003-05-11, will have metadata extracted as:

```
/v1/imagestore/xxx   hc:width     400

/v1/imagestore/xxx   hc:height    400

/v1/imagestore/xxx   hc:acquired  "2003-05-11"^^xsd:dateTime
```

The terms from `hc:` are hypothetical. For a file, the subject would be the file URL. More complex metadata structures will ideally involve the extension of the p-medicine ontology to accommodate them. This process effectively removes the need for user annotation in the semantic annotation tool, and such triples can be transferred to the main triplestore unchanged. An “identity” annotation description file will automatically be generated, and the

triples integrated as a result. Of course, this does not prevent a data manager or curator from changing the annotation description file later if there are problems or errors in the process.

3 User interface

The data warehouse user interface is a Liferay Portlet, Liferay being the chosen web portal interface for p-medicine. Once added to the Liferay portal, the front page of the Ampoule plugin is shown below:

The features of the interface are described below, from top to bottom. Note that the displayed interface is based upon fabricated datasets, since the tools required for semantic integration of real data were not finalised during development of this interface.

Free text search

Entering text here and clicking submit returns a list of triples where the subject, verb or object contain the given string, when converted to a string. So, for example, a triple with the verb "`has date of birth`" will be included in the results if "`date`" is entered, and a triple with the object "`54321`" will be included results if "`432`" is entered. This gives a simple Google style quick search of the warehouse contents.

Faceted browsing

Faceted browsing allows construction of queries based upon the most used classes and attributes in a database. As well as being displayed on the front page for all triples, when the faceted browsing facilities are also shown for results of queries. In this way, the user can browse and "drill down" into data, with more specific queries based upon results from previous ones.

3.1.1 Top classes

This displays a list of the most popular classes in the triples, that is, the object of triples where the verb is `rdf:type`. They are listed with links to display a list of the resources of that class – that is, the link to Patient will list patients, the link to Dataset will lists datasets. The number in brackets is the number of resources of that class. Only the top 5 most popular classes are shown, with a link to display a browsable list of the rest.

3.1.2 Top attributes

This is a list of the top used verbs in the triplestore. This allows browsing of resources by specific attributes. The link to the verb will list all triples with that verb. The link to specific values (objects) that are associated with that verb will display all triples with that verb and object. Only the top five most popular verbs are shown, with a link to display a browsable list of the rest.

Recent changes

This gives a list of recent changes to the data warehouse. This is a subset of the records in the warehouse access log. Log entries which are about querying or otherwise retrieving data are omitted for brevity and to offer some protection for researchers conducting novel analyses.

Warehouse news

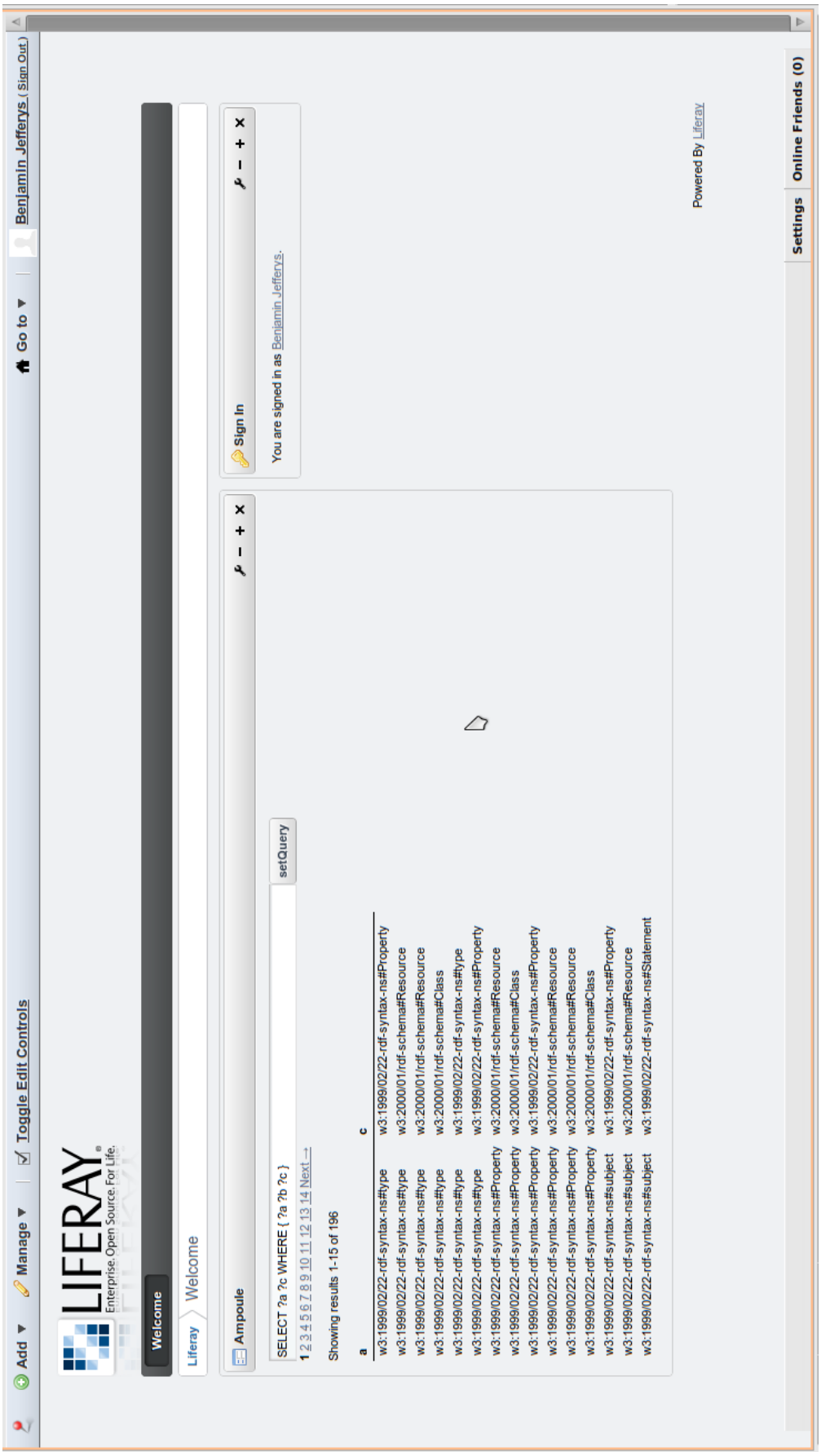
A list of important news stories from the warehouse administrators, including planned downtime and changes to interfaces.

Help, contact, terms and conditions

Links to help pages, contact information, and terms and conditions for using the data warehouse. This is yet to be written by the relevant legal and training work packages.

Advanced SPARQL query

This allows entry of a SPARQL query to query the triples in the triplestore:


Add
Manage
Toggle Edit Controls
Go to
Benjamin Jefferys (Sign Out)
LIFERAY
Enterprise. Open Source. For Life.
Welcome
Liferay
Welcome
Ampoule
SELECT ?a ?c WHERE { ?a ?b ?c }
setQuery
1 2 3 4 5 6 7 8 9 10 11 12 13 14 Next →
Showing results 1-15 of 196
a
c
w3:1999/02/22-rdf-syntax-ns#type
w3:2000/01/rdf-schema#Resource
w3:2000/01/rdf-schema#Class
w3:1999/02/22-rdf-syntax-ns#Property
w3:1999/02/22-rdf-syntax-ns#subject
w3:1999/02/22-rdf-syntax-ns#Statement
Sign In
You are signed in as Benjamin Jefferys.
Powered By Liferay
Settings
Online Friends (0)

Remaining work

Dedicated interfaces for auditing should be produced according to a requirements gathering exercise. Data commenting and rating, providing “social curation” of data, have also yet to be implemented.

4 Conclusion

The programmatic interface described in this document is deployed and in use by p-medicine partners integrating with the data warehouse. The user interface is not yet deployed in the form presented here because the faceted browsing interface and other interfaces rely on useful semantically annotated data being present. A simpler version is deployed on the p-medicine portal for SPARQL querying. Deployment of the newer interface is scheduled to happen after integration with the semantic annotator. Auditing and social commenting and rating features are future work for the user interface.